Machine Learning

Stanford University
Professor Andrew Ng

Jordan Hong

December 28, 2021

Contents

1 Introduction

1.1 What is Machine Learning

1. Machine Learning

   - Grew out of work in Artificial Intelligence (AI)
   - New capabilities for computers

2. Examples:

   - database mining
   - applications can't programby hand (handwriting recognition, Natural Language Processing (NLP), Computer Vision)
   - Neuromorphic applications

3. Definition

   - Arthur Samuel(1959)

     > Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.

   - Tom Mitchell(1998)

     > Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

4. Machine Learning in this course:

   (a) Suupervised Learning

   (b) Unsupervised Learning

   (c) Others: reinforcement learning, recommender systems

   (d) Practical application techniques

1.2 Supervised Learning

In supervised learning, the *the right answer* is given. For example:

1. Regression: predict real-valued output.

2. Classification: predict discrete-valued output.

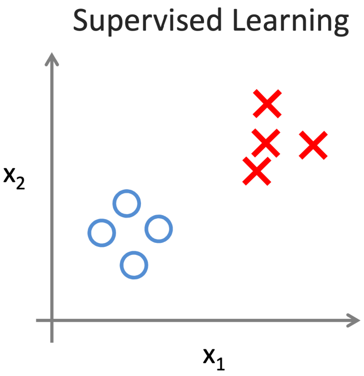


Figure 1: Supervised Learning

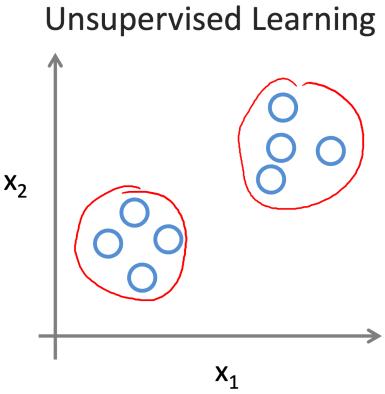


Figure 2: Unsupervised learning

1.3 Unsupervised Learning

The right answer is not given, e.g. cocktail problem (distinguishing two voices from an audio file.)

2 Linear Regression with One Variable

2.1 Model Representation

2.1.1 Notations

For a training set:

- **m** = Number of training examples.
- **x** = "input" variable / features.

- **y** = "output" variables / "target" variable.
- **(x,y)** - one training example.
- **($\mathbf{x^i}$,$\mathbf{y^i}$)** denotes the i^{th} training example

2.1.2 Hypothesis Function

A hypothesis function (h) maps input (x) to estimated output (y). How do we represent h?

$$\boxed{\textbf{Hypothesis Function} \quad h_\theta(x) = \theta_0 + \theta_1 x} \tag{1}$$

We can apply *Univariate linear regression* with respect to x.

2.2 Cost Function

Recall 1. The θ_is are parameters we have to choose. The intuition is is that we want to choose θ_i s such that h_θ is closest to y for our training examples (x,y).

$$\boxed{\textbf{Cost Function} \quad J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2} \tag{2}$$

Summary

1. **Hypothesis** $h_\theta(x) = \theta_0 + \theta_1 x$
2. **Parameters** θ_0, θ_1
3. **Cost Function** $J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$
4. **Goal** $\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$

2.3 Gradient Descent

2.3.1 Intuition

1. We have some function $J(\theta_0, \theta_1)$, we want to $\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$
2. Outline: start with some θ_0, θ_1, keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we end up at a minimum.

2.3.2 Gradient Descent Algorithm

Algorithm

repeat until convergence{

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0, \theta_1) \quad \text{(for j=0 and j=1).}$$

}

Notes

1. the := denotes non-blocking assignment, i.e. simultaneously updates $\theta_0 and \theta_1$
2. We use the derivative to find a local minimum.
3. α denotes the learning rate. Gradient descent can converge to a local minimum even when the learning rate α is fixed. As we approach a local minimum, gradient descent will automatically take smaller steps. Therefore it is not needed to decrease α over time.

2.3.3 Gradient Descent with Linear Regression

Recall, we have:

1. Gradient Descent Algorithm:

   repeat until convergence{

   $$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad \text{(for j=0 and j=1).}$$

   }

2. Linear Regression Model:

   $$h_\theta(x) = \theta_0 + \theta_1 x$$

   $$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

   We can substitute the above equations, which gives us:

   $$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

   $$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

3 Review of Linear Algebra

This is section is a basic review of linear algebra. I have skipped this section for now and will come back to it if time permits.

4 Linear Regression with Multiple Variables

4.1 Multiple features

Recall in the single variable case, we have a single input (x), two parameters(θ_0, θ_1). The hypothesis can be expressed as:

$$h_\theta(x) = \theta_0 + \theta_1 x.$$

Now, consider a generalized case where there are multiple features: X_1, X_2, X_3. The information can be organized in a table with example numerical values:

Sample Number (i)	X_1	X_2	y
1	6	87837	787
2	7	78	5415
3	545	778	7507
4	545	18744	7560
5	88	788	6344

Table 1: Sample Table

From Table 1, one can see that each row is a sample a feature on each column.

4.1.1 Notation

1. **n**: number of features.
2. $\mathbf{x^{(i)}}$: (row vector) input features of the i^{th} training example. i= 1, 2,. . . , m.
3. $\mathbf{x^{(i)}}_{\mathbf{j}}$: value of feature j in the i^{th} training example. j= 1, 2, . . . , n.

4.1.2 Hypothesis

Previously,

$$h_\theta(x) = \theta_0 + \theta_1 \cdot x$$

Now, we can extend the hypothesis to :

$$h_\theta(x) = \theta_0 \cdot 1 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2$$

For convenience of notation, let's define x_0=1, i.e. $\mathrm{x}^{\mathrm{i}}{}_0$=1 ∀ i.

Therefore, we have: $\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ and $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$. Then, the hypothesis function can be written as:

$$\begin{aligned} h_\theta(x) &= \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \dots & \theta_n \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\ &= \theta^T \cdot \mathbf{x} \end{aligned} \tag{3}$$

This is *Multivariate linear regression.*

4.2 Gradient Descent for Multiple Variables

4.2.1 Algorithm

Summary for Multivariables

1. **Hypothesis** $h_\theta(x) = \theta^T \cdot \mathbf{x}$
2. **Parameters** θ
3. **Cost Function** $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
4. **Goal** $\min_\theta J(\theta)$

Gradient Descent for Multiple Variables

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \qquad \text{(for j=0, 1,... n)}$$

}

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(\mathbf{x}^{(\mathbf{i})}) - y^{(i)}) \cdot x_j^{(i)}$$

Note: $x_0^{(i)} = 1$, by definition.

4.2.2 Vectorized Implementation

One can work out the linear algebra and arrive at the following simplification using vectorized operations. The cost function J can be expressed as:

$$J(\theta) = \frac{1}{2m}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) \tag{4}$$

The MATLAB implementation is as follows:

```
m = length(y); % calculate how many samples
J = 1/(2*m)*((X*theta—y).')*(X*theta—y);
```

Gradient descent can be vectorized in the form:

$$\theta = \theta - \frac{\alpha}{m} \cdot \mathbf{X}^T \cdot (\mathbf{X}\theta - \mathbf{y}) \tag{5}$$

The MATLAB implementation is as follows:

```
m = length(y); % number of training examples
for iter = 1:num_iters
    theta = theta — alpha/m* X.'*(X*theta —y);
```

4.3 Gradient Descent in Practise I: Feature Scaling

- Idea: ensure each featurre are on a similar scale
- Get every feature into approx. $-1 \leq x_i \leq 1 (\sim$ order)
- **Mean Normalization**: Replace x_i with $\frac{x_i - \mu_i}{s_i}$, where μ_i and s_i are the sample mean and standard deviation, respectively.

4.4 Gradient Descent in Practise II: Learning Rate

- Ensure gradient descent is working: plot J_θ over each number of iteration (not over θ !)
- Example automatic convergence test: for sufficiently small α J_θ should decrease by less than 10^{-3} i one iteration.
- If α is too small, gradient descent can be slow to converge.
- If α is too large, gradient descent may not converge.
- To choose α, try 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1... (by 3x)

4.5 Features and Polynomial Regression

We can fit into different polynomials by choice, using multivariate regression. Recall

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

Let x_1 be x^1, x_2 be x^2, x_3 be x^3. Note we should still apply feature scaling to x_1, x_2, and x_3 individually!

4.6 Normal Equation

The normal equation provides a method to solve for θ analytically. For our data with m samples, n features, recall each sample can be written as:

$$\mathbf{x^{(i)}} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_j^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$$

We can construct a design matrix:

$$\mathbf{X} = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ — & (x^{(3)})^T & — \\ & \vdots & \\ — & (x^{(m)})^T & — \end{bmatrix} \tag{6}$$

Then θ can be found by the normal equation:

$$\theta = (\mathbf{X^T X})^{-1}\mathbf{Ty} \tag{7}$$

Normal equation is useful as no α is required to and we do not need to iterate. However, we do have to compute $(X^TX)^{-1}$, which can be computationally expensive when n is large. The complexity is O(n^3) for inverse operations. Gradient Descent is useful when n is large (many features).

4.7 Normal Equation and Non-invertibility

What if $(X^TX)^{-1}$ is non-invertible?

- Redundant features (linearly dependent), i.e. having same information in two different units.
- Too many features (i.e. m $\leq$ n). Delete some features or use regularization

5 Octave/MATLAB Tutorial

6 Logistic Regression

6.1 Classification

- Binary Classification: y $\in \{0, 1\}$, where 0 denotes the negative class; 1 denotes the positive class.

- Multi-class Classification:y $\in \{0, 1, \cdots, n\}$

We will be using binary classification:

- Linear regression is not suitable for classification: since $h_\theta(x)$ can output out of range, i.e. < 0 or > 1.
- We will use **logistic regression**, which ensures that the output $h_\theta(x)$ is between 0 and 1.

6.2 Hypothesis Representation

6.2.1 Logistic function

The idea is to have $0 \leq h_\theta(x) \leq 1$. Instead of the linear regression hypothesis: $h_\theta(x) = \theta^T x$, we will let:

$$h_\theta(x) = g(\theta^t x),$$

where

$$g(z) = \frac{1}{1 + e^{-z}}$$

g(z) is known as the **logistic function**, also known as the Sigmoid function. Figure 3 shows a plot of the logistic function, which ranges from 0 to 1.

which yields

$$h_\theta(x) = \frac{1}{1 + e^{-z}} \tag{8}$$

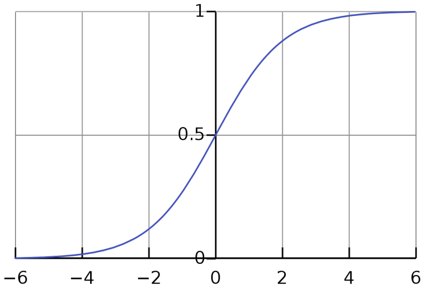


Figure 3: Sigmoid Function

6.2.2 Interpretation of Hypothesis Output

$h_\theta(x)$ = estimated probability that y= 1 on input x. For example, $h_\theta(x) = 0.7$ gives us a probability of 70% that are output is 1. From a probability theory point of view, one can express $h_\theta(x)$ as $P(y = 1 \mid x; \theta)$.

Note that since this is a probability and the total probability sums up to 1, and the real y can only be either 0 or 1:

$$P(y = 0 \mid x; \theta) + P(y = 1 \mid x; \theta) = 1$$

6.3 Decision Boundary

Recall so far we have $h_\theta(x) = g(\theta^T x)$ and $g(z) = \frac{1}{1+exp(-z)}$. Suppose we set $h_\theta(x) = 0.5$ to be our determining factor for whether y= 0 or y =1. Note that from 3, one can observe that $h_\theta(x) = 0.5$ corresponds to $\theta^T x = 0$, which is the **decision boundary**. The decision boundary is the equation which separates the different classes on a plot. There are linear and non-linear decision boundaries.

6.4 Cost function

Previously, we had $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$, where $Cost(h_\theta(x), y) = \frac{1}{2}(h_\theta(x) - y)^2$. Now, the definition of the hypothesis h_θ has changed to $\frac{1}{1+exp(-\theta^T x}$, as a result the cost function is now non-convex.

Logistic Regression Cost Function
Therefore, a new cost function definition is needed. We propose:

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note that Cost=0, if y=1, $h_\theta(x) = 1$; but as $h_\theta(x) \to 0$, then $Cost \to \infty$. This proposition captures the intuition that if $h_\theta(x) = 0$, predict $P(y = 1 \mid x; \theta)$, but y ends up being 1, we will penalize the learning algorithm by very large cost.

6.5 Simplified Cost Function and Gradient Descent

Since y can only be either 0 or 1, we can simplify the cost function.

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m} Cost\,(\,h_\theta(x^{(i)})\,,\,y^{(i)}) \\ &= \frac{-1}{m}\,[\,\sum_{i=1}^{m} y^{(i)}\,log\,h_\theta(x^{(i)})\; + \;(1 - y^{(i)})\,log\,(\,1 - h_\theta(x^{(i)})\,)\,] \end{aligned} \tag{9}$$

Equation 9 is based on Maximum Likelihood Estimation.

A vectorized implementation is, for a design matrix $\mathbf{X}$:

$$\boxed{h = g(\mathbf{X}\theta)} \tag{10}$$

$$\boxed{J(\theta) = \frac{1}{m}(-y^T log(h) - (1 - y)^T log(1 - h))} \tag{11}$$

For gradient descent,we would want to $\min_\theta J(\theta)$:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

We can compute the partial derivative of $J(\theta)$, which is identical to that of linear regression:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

However, in this case the hypothesis function $h_\theta(x) = \frac{1}{1+exp(-\theta^T x}$ has changed!

A vectorized implementation for this is:

$$\boxed{\theta := \theta - \frac{\alpha}{m} \mathbf{X}^T (g(\mathbf{X}\theta) - \mathbf{y}))} \tag{12}$$

6.6 Advanced Optimization

6.6.1 Taking a Step Back

If we take a step back, and consider essentially what tasks we are performing. We need to compute two things:

1. $J(\theta)$
2. $\frac{\partial}{\partial \theta_j} J(\theta)$

6.6.2 Optimization Algorithm

There exists other more sophisticated and faster ways to optimize θ instead of gradient descent; they often do not involve selecting learning rate α and are more efficient. However, these algorithms are harder to code by hand. It is suggested that we use libraries for such algorithms.

We can write a single function that returns both $J(\theta)$ (jVal) and $\frac{\partial}{\partial \theta_j} J(\theta)$ (gradient):

```
function [jVal, gradient] = costFunction(theta)
    jVal = [...code to compute J(theta)...];
    gradient = [...code to compute derivative of J(theta)...];
end
```

Then we can use octave's "fminunc()" optimization algorithm along with the "optimset()" function that creates an object containing the options we want to send to "fminunc()".

```
    options = optimset('GradObj', 'on', 'MaxIter', 100);
    initialTheta = zeros(2,1);
    [optTheta, functionVal, exitFlag] = fminunc(@costFunction,
        initialTheta, options);
```

We then give to the function "fminunc()" our cost function, our initial vector of theta values, and the "options" object that we created beforehand.

6.7 Multiclass Classification: One-vs-All

Now, let's extend the binary classification of data to multi-classes, i.e expanding our definition of y s.t. y={0, 1, ..., n}. We will divide out problem into n+1 (0...n) binary classification problems. In each problem, we predict the probability that y is a memember of one of our class. We train a logistic regression classifier $h_\theta^{(i)}(x)$ ∀ i to predict y = i:

$$h_\theta^{(i)}(x) = P(y = i \mid x; \theta) \tag{13}$$

Figure 4 shows an example of the procedure of classifying three classes. We choose one class and then lump all the others into a single second class (hence the name One-vs-All). We apply the binary logisic regression repeatedly and use the hypothesis that returns the highest value.

$$prediction = \max_i \left(h_\theta^{(i)}(x) \right) \tag{14}$$

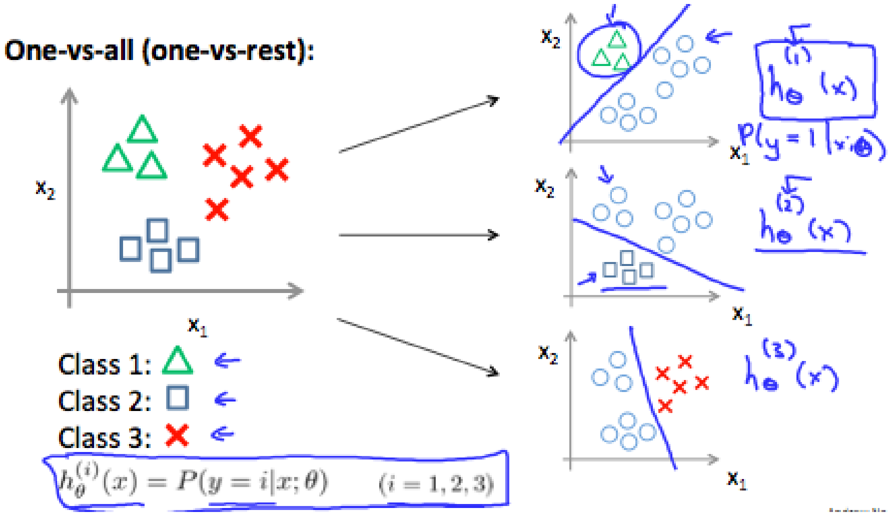


Figure 4: Example of Multiclass Logistic Regression

7 Regularization

7.1 The problem of overfitting

There are two types of errors in fitting functions to data, in which data shows the structure is not captured by the model. The terminology applies to both linear and logistic regression. Figure 5 shows an example of each fitting case.

1. Underfitting (high bias): when the form of our hypothesis function ($h_\theta(x)$) maps poorly to the trend of data. It is usually caused by a function that is either too simple or uses too little features.

2. Overfitting (high variance): when hypothesis function learns the training set very well hence fits the available data but does not generalize well to predict new data, i.e fitting too many details. It is usually caused by a complicated function that creates a lot of unneccessary curves and angles unrelated to the data.

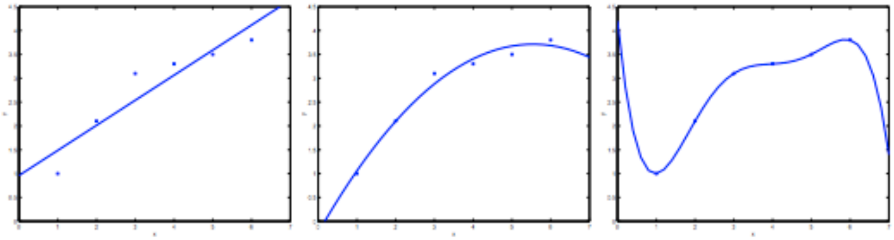


Figure 5: Three scenarios of fitting: underfit(left), good fit (middle), and overfit(right).

There are two main options to address the issue of overfitting:

1. Reduce the number of features
   - manually select which features to keep.
   - Use a model selection algorithm (later in the course).

2. **Regularization**
   - Keep all the features, but reduce the magnitude of parameters θ_j
   - Regularization worls well when we have a lot of slightly useful features. (Each contributes a bit to predict y)

7.2 Cost Function

7.2.1 Intuition

Suppose we have overfitting, we can reduce the weight that some of the terms in our function carry by penalizing the feature parameter with increased cost. Consider the following hypothesis function :

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta x^3 + \theta x^4.$$

We would want to make the function more quadratic, which means we would like to reduce the influence of θ_3 and θ_4. Since our goal was to minimize the cost function $J(\theta)$, we could modify the original cost function

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000 \cdot \theta_3^2 + 1000 \cdot \theta_4^2.$$

This will force θ_3 and θ_4 to be close to zero, and make the original hypothesis function more quadratic.

7.2.2 Regularization

Now, we can take a step further and regularize all theta parameters:

$$\min_\theta \, [\frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2] + \lambda \sum_{j=1}^{n} \theta_j^2 \tag{15}$$

Here we are performing two separate operations which can be oberved as the two terms in Equation 15. The first term corresponds to "training the data", and the second term relates to "keeping the parameters small". The coefficient λ is the **regularization parameter** and determines the balance between the aforementioned two objectives. Note that if λ is too big ($\approx 10^{10}$), then all $\theta_j \approx 0$, except for θ_0 (the constant term). This makes $h_\theta \approx 0$ and defies the purpose of training and leads to underfitting (a flat horizontal line).

7.3 Regularized Linear Regression

7.3.1 Gradient Descent

We will modify the gradient descent function to separate out θ_0 from the rest of the parameters because we do not want to penalize θ_0.

repeat until convergence{

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha [(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}) + \frac{\lambda}{m} \theta_j]$$

$\}$, where $j \in \{1, 2, \ldots, n\}$

The above equation can be re-arranged to get the final form:

$$\boxed{\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}} \tag{16}$$

The term $(1 - \alpha\frac{\lambda}{m})$ will always be less than 1. Intuitively, this reduces the parameter, then the rest of equation 16 carries the normal gradient descent.

7.3.2 Normal Equation

Recall from our previous normal equation (Equation 7), we have the design matrix $\mathbf{X}$ such that

$$\mathbf{X} = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ — & (x^{(3)})^T & — \\ & \vdots & \\ — & (x^{(m)})^T & — \end{bmatrix}$$

Now, to add in regularization to Equation 7, we get:

$$\boxed{\theta = ((\mathbf{X^T X}) + \lambda \cdot \mathbf{L})^{-1}\mathbf{x}^T y} \tag{17}$$

where $L \in \mathbb{R}^{(n+1)\times(n+1)}$

$$L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix} \tag{18}$$

Supopose non-invertibility arises: $m \leq n$, i.e #examples is less than or equal to number of features, then using a $\lambda > 0$ in Equation 17 will resolve the non-invertibility.

7.4 Regularized Logistic Regression

7.4.1 Regularized Logistic Cost Function

We can modify our old logistic cost function (Equation 9) to apply regularization.

$$\boxed{J(\theta) = \frac{-1}{m}\sum_{i=1}^{m}[y^{(i)}\,log\,h_\theta(x^{(i)}) \;+\; (1 - y^{(i)})\,log\,(\,1 - h_\theta(x^{(i)})\,)] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2} \tag{19}$$

Note that the second sum in Equation 19 explicitly exludes the bias term θ_0.

7.4.2 Regularized Logistic Gradient Descent

The gradient descent is similar to that of linear regression; however note that the hypothesis function has been updated to Equation 8.

repeat until convergence{

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha[(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}) + \frac{\lambda}{m}\theta_j]$$

}

8 Neural Networks: Representation

8.1 Non-linear Hypothesis

- Example: non-linear classification- logistic regression has lots of terms and not scalable($O(n^2), O(n^3)$).
- Application: computer vision
- Neural network is useful for **non-linear, large scale classification**.

8.2 Neurons and the brain

- Origin: Neuromorphic computing mimics how brain functions.
- Recent resurgence due to hardware advancement.
- The “one-learning hypothesis”: there exists a single algorithm that the brain uses for generalized learning.
  - Neural re-wiring: wiring the auditory cortex to eyes, then the cortex learns how to *see*.

8.3 Model representation

Figure 6 shows a model of a biological neuron. The **dendrites** are the wires which receive input signal. The **axons** output the signal.

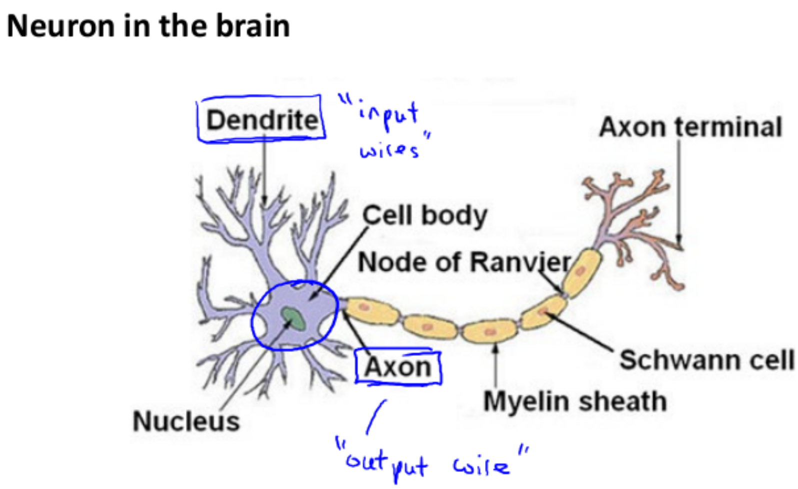


Figure 6: A biological neuron

8.3.1 Neural model: logistic unit

In our model:

- Dendrites: input features x_1, x_2, ..., x_n.
- Output: result of hypothesis function.
- x_0: bias unit (:= 1).
- Logistic function: $\frac{1}{1+exp(-\theta^T X)}$, referred to as the sigmoid(logistic) **activation** function.
- θ are referred to as **weights**.

Visually, a simplistic representation can be view as

$$[x_0\ x_1\ x_2] \rightarrow [\ ] \rightarrow h_\theta(x) \tag{20}$$

The input nodes (layer 1, input layer) go into the another node (layer 2), which then outputs to hypothesis function (output layer).

In this example, we label the intermediate layers of nodes (between the input and output layers) $a_0^2, \ldots, a_n^2$: **activation units**. Definitions:

$$\boxed{a_i^{(j)} = \text{“activation of unit i in layer j}}$$

$$\boxed{\Theta^{(j)} = \text{matrix of weights controlling function mapping from layer j to layer j+1.}}$$

If we had one hidden layer, it would look like:

$$\boxed{[x_0\, x_1\, x_2\, x_3] \;\rightarrow\; [a_1^{(2)}, a_2^{(2)}, a_3^{(2)}] \;\rightarrow h_\theta(x)}$$

The value for each of the activation node is obtained as follows;

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(1)}a_0 + \Theta_{11}^{(1)}a_1 + \Theta_{12}^{(1)}a_2 + \Theta_{13}^{(1)}a_3)$$

Essentially, each layer j has its own **matrix of weights**, $\Theta^{(j)}$.

If network has s_j units in layer j and s_{j+1} units in layer j+1, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

The $+1$ comes from the addition in $\Theta^{(j)}$ of the bias nodes, x_0 and $\Theta_0^{(j)}$.

In the above example, layer 2 has 3 units ($a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$) and layer 1 has 3 units (not including the bias unit). Therefore the weight matrix $\Theta^{(1)}$ which maps layer 1 to layer 2 is of dimension $3 \times (3 + 1)$.

Another example: if layer 1 has 2 input nodes and layer 2 has 4 activation nodes- dim ($\Theta^{(1)}$) is going to be 4x3 where $s_j = 2$ and $s_{j+1} = 4$, so $s_{j+1} \times (s_j + 1) = 4 \times 3$.

8.3.2 Forward Propagation: Vectorized Implementation

Define a new variable: $z_k^{(j)}$ as the parameter for for the logistic function $g(z)$ such that:

$$a_1^{(2)} = g(z_1^{(2)})$$
$$a_2^{(2)} = g(z_2^{(2)})$$
$$a_3^{(2)} = g(z_3^{(2)})$$

Therefore, for layer j=2 and unit k:

$$z_k^{(2)} = \Theta_{k,0}^{(1)}x_0 + \Theta_{k,1}^{(1)}x_1 + \cdots + \Theta_{k,n}^{(1)}x_n$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \; z^{(j)} = \begin{bmatrix} z_1^{(j)} \\ z_2^{(j)} \\ \cdots \\ z_n^{(j)} \end{bmatrix}$$

We can further generalize by setting $\mathbf{x} = a^{(1)}$ and re-write the equation:

$$z^{(j)} = \Theta^{(j-1)}a^{(j-1)} \tag{21}$$

Dimension check:

- $\Theta^{(j-1)}$ is of dimension $s_j \times (n+1)$.
- a$^{(\mathrm{j})}$ is a column vector with dimension $1 \times (j+1)$.

Thus the logistic function is applied element-wise to **z**:

$$a^{(j)} = g(z^{(j)}) \tag{22}$$

We then add the bias unit $a_0^{(j)} = 1$ to the activation matrix. If we propagate forward by one layer in Equation 21:

$$z^{(j+1)} = \Theta^{(j)} a^{(j)} \tag{23}$$

The last theta matrix $\Theta^{(j)}$ will have only one row which is multiplied by one column $a^{(j)}$ such that the result is a single scalar.Hence the final hypothesis output is:

$$h_\theta(x) = a^{(j+1)} = g(z^{(j+1)}) \tag{24}$$

8.4 Application - Logic Gate Synthesis

We can apply neural networks to predict various logic gates: AND, OR, XOR, NOR. The basic graph can be represented as :

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \; [\; g(z^{(2)}) \;] \rightarrow \; h_\Theta(x)$$

8.4.1 AND Gate

Let the first theta matrix be:

$$\Theta^{(1)} = [\; -30 \;\; 20 \;\; 20 \;]$$

Table 2 outlines the logistic hypothesis output which represents an AND gate: output $= 1 \iff (\; x_1 = 1$ and $x_2 = 1)$. The intuition here in choosing the Θ parameters is that we want the sigmoid function evaluate to be positive only when both x$_1$ and x$_2$ are logic 1. Therefore, we assign the bias unit to a large negative value, such that the result of Θx is positive only when both the two other weights are added.

8.4.2 OR Gate

Similarly, Table 3 outlines the truth table for the OR function. We pick a parameter matrix:

$$\Theta^{(1)} = [\; -10 \;\; 20 \;\; 20 \;]$$

x_1	x_2	z	$h_\Theta(x) = g(z)$
0	0	-30	0
0	1	-10	0
1	0	-10	0
1	1	10	1

Table 2: Truth table of AND gate

x_1	x_2	z	$h_\Theta(x) = g(z)$
0	0	-10	0
0	1	10	1
1	0	10	1
1	1	30	1

Table 3: Truth table of OR gate

8.4.3 NOR Gate

Similarly, Table 4 outlines the truth table for the NOR function. We pick a parameter matrix:

$$\Theta^{(1)} = [\ 10 \quad -20 \quad -20\ ]$$

8.4.4 NOT Gate

For an inversion, we just need a single parameter x. Table 5 outlines the truth table for the NOT function. We pick a parameter matrix:

$$\Theta^{(1)} = [\ 10 \quad -20\ ]$$

8.4.5 XNOR Gate

Now, we would like to synthesize the XNOR function (shown in Table 6. This requires a three level neural network. We can decompose the function XNOR:

$$\overline{x_1 \oplus x_2} = x_1 x_2 + \overline{x_1} \cdot \overline{x_2}$$

x_1	x_2	z	$h_\Theta(x) = g(z)$
0	0	10	1
0	1	-10	0
1	0	-10	0
1	1	-30	0

Table 4: Truth table of NOR gate

x	z	$h_\Theta(x) = g(z)$
0	10	1
1	-10	0

Table 5: Truth table of NOT gate

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Table 6: Truth table of XNOR

Therefore, in the second layer we will computer the AND and NOR functions then apply OR to the two intermediate output in the third layer.

$$\Theta^{(1)} = \begin{bmatrix} -30 & 30 & 20 \end{bmatrix}$$

$$\Theta^{(2)} = \begin{bmatrix} -10 & 20 & 20 \end{bmatrix}$$

So:

$$a^{(2)} = g(\Theta^{(1)} \cdot x)$$
$$a^{(3)} = g(\Theta^{(2)} \cdot a^{(2)})$$
$$h_\Theta(x) = a^{(3)}$$

8.5 Multi-class Classification

For multi-class classification, our prediction will be a column vector with each element be of “whether the current item belongs to the class of that position”. For example, for a 4-class classification, an example output would be:

$$h_\Theta(x) = [0010]$$

9 Neural Networks: Learning

9.1 Cost Function

Define variables:

- L :total number of layers in the network.
- s$_l$: number of units (not counting bias unit) in layer l.
- K: number of output classes.

Recall for classifications, we have binary and multi-class:

1. Binary classification:
   - y = 0 or 1.
   - 1 output unit, i.e. $h_\Theta(x) \in \mathbb{R}$.
   - S$_L$ = 1; K = 1.
2. Multi-class classification:
   - $y \in \mathbb{R}^K$
   - S$_L$= K. K≥3.

Recall the cost function for regularized logistic regression in Equation 19, shown below:

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} [y^{(i)} \, log \, h_\theta(x^{(i)}) \; + \; (1 - y^{(i)}) \, log \, (\, 1 - h_\theta(x^{(i)}) \,)] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

For neural networks, we will extend Equation 19 into a generalized form:

$$\begin{aligned} J(\Theta) = & \frac{-1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} [y_k^{(i)} \, log \, h_\theta(x^{(i)})_k \; + \; (1 - y_k^{(i)}) \, log \, (\, 1 - h_\theta(x^{(i)})_k \,)] \\ & + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{j,i}^{(l)})^2 \end{aligned} \tag{25}$$

In the first part of Equation 25, we added a summation to account for multiple output nodes. Note:

- Double sum adds up logistic regression costs calculated for each cell in the output layer.
- Triple sum adds up the squares if all individuals Θs in the entire network.
- The i in the triple sum does not refer to the training example i !

9.2 Backpropagation Algorithm

- Backpropagation: neural-network terminology for minimizing cost function.
- Gradient Computation: need to computer $J(\Theta)$ and $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Given a training set $\{ (x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})\}$. Set $\Delta^{(l)}_{i,j} = 0$ $\forall$ l,i,j.

For all training examples t=1:m:

1. Set $a^{(1)} := x^{(t)}$
2. Perform forward propagation to computer all the nodes ($a^{(l)}$).
3. Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$. (Vectorized deviation of output units to correct values).
4. Backpropagate the error:

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$$

   This equation makes use of the fact that the derivative of the sigmoid function:

$$g'(z^{(l)}) = a^{(l)} .* (1 - a^{(l)})$$

5. Obtain the estimation of gradient ($\nabla J(\Theta)$). Intuitively, $\delta^{(l)}$ is the error for the activation unit in layer l: a^{l}. more formally, the delta values are the derivatives of the cost functions.
   It can be shown that

$$\boxed{\frac{\partial}{\partial \Theta^{(l)}_{i,j}} J(\Theta) = a^{(l)}_j \cdot \delta^{(l+1)}_i}$$

   Hence, a vectorized implementation of the accumulated gradient is:

$$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$

We then normalize and add regularization to obtain the gradient:

- $D^{(l)}_{i,j} := \frac{1}{m} (\Delta^{(l)}_{i,j} + \lambda \Theta^{(l)}_{i,j})$, if j $\neq$ 0
- $D^{(l)}_{i,j} := \frac{1}{m} \Delta^{(l)}_{i,j}$, if j=0

And thus $\frac{\partial}{\partial \theta^{(l)}} = D^{(l)}$.

9.3 Gradient checking

- Purpose: Eliminates error from escaping. Assuring backpropagation is working as intended.
- Approx derivative as:

$$\frac{\partial}{\partial \Theta_j} J(\Theta) \approx \frac{J(\Theta_1, \dots, \Theta_j + \epsilon, \dots, \Theta_n) - J(\Theta_1, \dots, \Theta_j - \epsilon, \dots, \Theta_n)}{2\epsilon}$$

- Compute the gradient approximation and compare with backpropagation(delta vector)
- Turn off gradient checking as this is computationally expensive.

Below is an implementation of gradient checking in MATLAB code:

```
epsilon = 1e-4;
for i = 1:n,
  thetaPlus = theta;
  thetaPlus(i) += epsilon;
  thetaMinus = theta;
  thetaMinus(i) -= epsilon;
  gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*epsilon)
end;
```

9.4 Random Initialization

- Initializing all theta weights to zero does not work. In backpropagation, all nodes will update to the same value repeatedly, creating **symmetry**.
- Symmetry breaking: initialize to values that range in [-ϵ, ϵ]

10 Advice for Applying Machine Learning

10.1 Deciding What to Try Next

Potential next steps:

1. Larger training data.
2. Smaller set of feature.
3. Additional features.
4. Polynomial features.
5. Increase λ.
6. Decrease λ.

Machine Learning Diagnostic: a test to run to gain insight on feasibility of techniques and how to optimize performance.

10.2 Evaluating Hypothesis: training/validation/test sets

10.2.1 Motivation with Test sets

We want to address the problem of overfitting: fits training examples very well (low error) but fails to generalize and hence not inaccurate on additional data. Thus, to evaluate the true accuracy of hypothesis, we randomly split the data into two sets: a **training set** (70%) and a **test set** (30%). The procedure :

1. Learn Θ and minimize $J_{train}(\Theta)$ using the training set
2. Compute test set error: $J_{test}(\Theta)$.

   (a) For linear regression:

   $$J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

   (b) For logistic regression:

   $$J(\theta) = \frac{-1}{m_{test}} \sum_{i=1}^{m_{test}} (y_{test}^{(i)} log(h_\Theta(x_{test}^{(i)}) + (1 - y_{test}^{(i)}) log(1 - h_\Theta(x_{test}^{(i)}))$$

   (c) For classification, we first calculate the misclassification error.

   $$err(h_\Theta(x), y) = \begin{cases} 1 & \text{if } (h_\Theta(x) \geq 0.5 \;\&\; y = 0) \text{ or } (h_\Theta(x) \leq 0.5 \;\&\; y = 1) \\ 0 & \text{if otherwise.} \end{cases} \tag{26}$$

   The average test error is thus:

   $$\text{Test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\Theta(x), y) \tag{27}$$

10.2.2 Polynomial features: Train/Validation/Test model

- Computed cost is less than the generalization error: We trained our data on the training set to obtain Θ with different degrees of polynomial (d). We then choose d based on minimizing the test set.
- Problem: $J_{test}(\Theta)$ is likely to be an optimistic generalization error, since the parameter d is fit to the test data set.
- Solution: break down dataset into three sections:
  1. Training set (60%).
  2. Cross-validation set (20%).
  3. Test set (20%).

Procedure:

1. Optimize the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree d with the least error using the cross validation set.
3. Estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$, (d = theta from polynomial with lower error);

This way, the degree of the polynomial d has not been trained using the test set.

10.3 Diagnosing Bias vs Variance

- Degree of polynomial (d) and underfitting (high bias) / overfitting (high variance).
- Training errors tend to decrease as d increases.
- Cross validation error tends to form a convex curve with a minimum point.

Therefore, the key to distinguish between bias and variance is summarized in Figure 7.

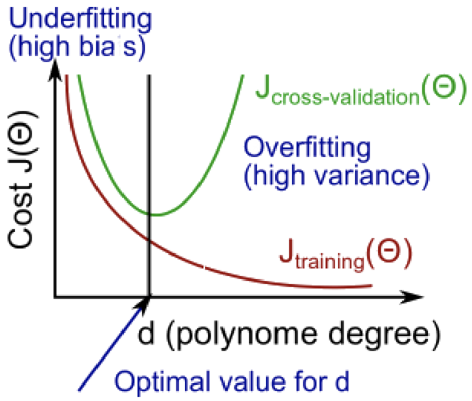


Figure 7: Error cost of cross-validation and training over degree of polynomial

1. **High bias (underfitting)**: both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ will be high. Thus: $J_{train}(\Theta) \approx J_{CV}(\Theta)$
2. **High variance (overfitting)**: *Low* $J_{train}(\Theta)$ and *high* $J_{CV}(\Theta)$. Thus: $J_{train}(\Theta) << J_{CV}(\Theta)$

10.4 Regularization and bias and variance

- Large λ (lower significance of Θ, heavy penalization so $h_\theta \approx 0$): high bias, underfit.
- Small λ (high significance of Θ): high variance, overfit.

Procedure to pick the right λ:

1. Create list of lambda.
2. Iterate through λs and learn/obtain Θ.
3. Compute cross-validation error ($J_{CV}(\Theta)$) using the learned Θ **without regularization** (i.e. $\lambda = 0$)
4. Select the best lambda and Theta pair and verify on test set. (Compute $J_{test}(\Theta)$.

10.5 Learning Curves

Plotting error over number of training examples.

10.5.1 High bias

1. Low training set size: low J_{train} and high J_{CV}.
2. Large training set size: high J_{train} and high J_{CV}.

Getting more training data will not help.

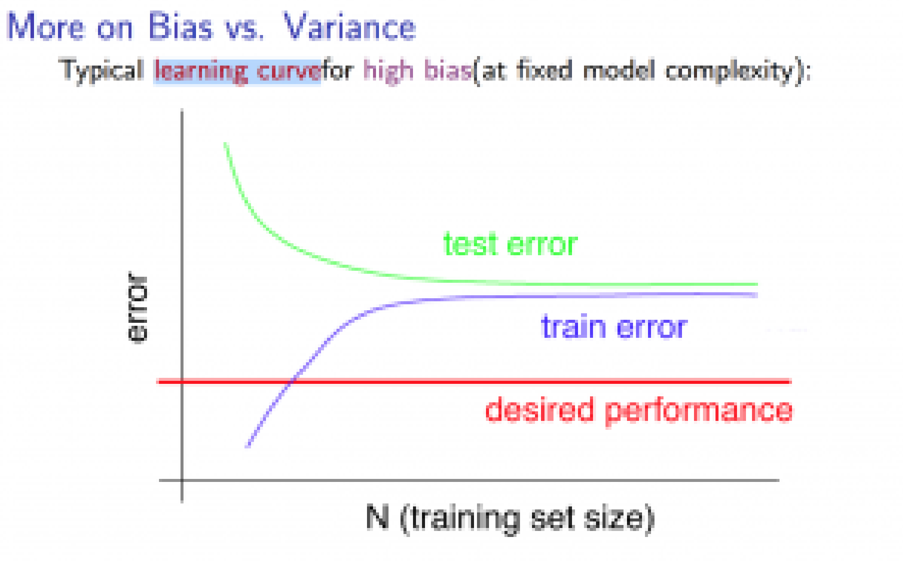
10.5.2 High variance

1. Low training set size: low J_{train} and high J_{CV}.
2. Large training set size: J_{train} increases and J_{CV} decreases.

Getting more training data will help.

10.6 Summary

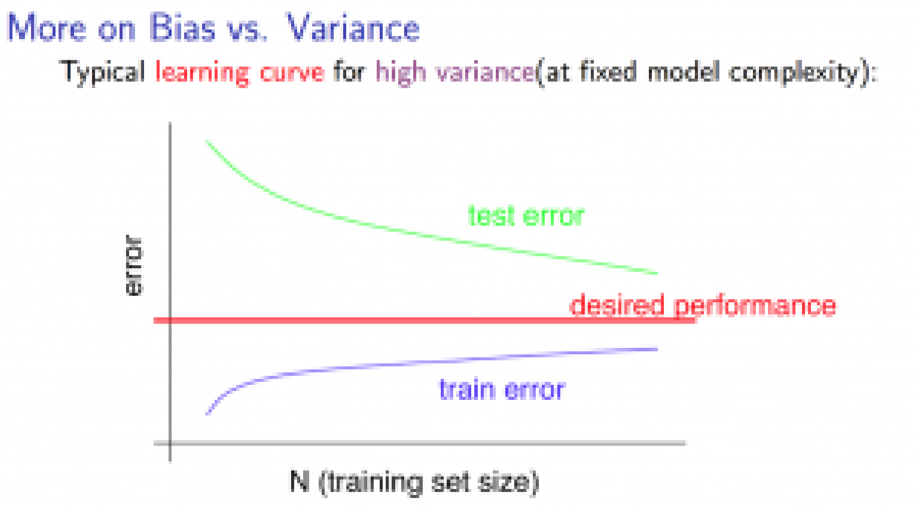
1. Getting more training examples: Fixes high variance
2. Trying smaller sets of features: Fixes high variance
3. Adding features: Fixes high bias
4. Adding polynomial features: Fixes high bias
5. Decreasing λ: Fixes high bias
6. Increasing λ: Fixes high variance.



More on Bias vs. Variance

Typical learning curvefor high bias(at fixed model complexity):

Figure 8: Learning curve for high bias



More on Bias vs. Variance

Typical learning curve for high variance(at fixed model complexity):

Figure 9: Learning curve for high variance

10.7 Neural Networks

1. Small neural network: fewer parameters $\implies$ prone to underfitting, but computationally cheaper.
2. Large neural network: more parameters $\implies$ prone to overfitting, computationally expensive. Use regularization to address overfitting.

11 Machine Learning System design

11.1 Prioritizing what to work on: Spam Classification Example

- Supervised learning.
- $\mathbf{x}$ = features of email (words indicative of spam).
- y = 1 (spam) or 0 (not spam).
- In practise, we take the most frequent n words (10000 to 50000) in training set to use as elements in x.

Potential next steps:

1. Collect lots of data.
2. Develop sophisticated features, e.g. for email routing information, message body, etc.
3. Sophisticated algorithms.

11.2 Error Analysis

11.2.1 Recommended Approach

- Start with simple algorithm that can be implemented quickly. Implement and test on *Cross-validation data*.
- Plot learning curves to decide if more data or more features will help.
- Error analysis: Manually examine errors from the examples in *cross-validation* set. Observe any systematic trend.

11.2.2 Error Analysis

Questions to ask:

1. What type of email it is.
2. What cues (features) do you think would have helped the algorithm to classify them correctly.

11.2.3 The importance of numerical evaluation

- Should discount/discounts/discounted/discounting be treated as the same word?
    - Use "stemming" software, e.g. *Porter Stemmer*
- Error analysis might not help. Instead, try and see it it works.
- Need metric (numerical evaluation) of algorithm to evaluate performance. E.g. cross validation error.

11.3 Error Metrics for Skewed Classes

11.3.1 Motivation

Consider case for cancer classification. If we get 1% error on the test set, i.e. 99% diagnoses are correct. However, only 0.5% of patients have cancer in the first place. Is the 1% error a good evaluation? Furthermore, consider the example prediction algorithm:

```
function y = predictCancer(x)
    y=0; %ignore x!
return
```

Regardless of the dataset, the hypothesis will always predict y=0.

11.3.2 Precision/Recall

	Actual	1	0
Prediction	1	True positive	False positive
	0	False negative	True negative

Table 7: Truth Table

We can thus define two measures of accuracy:

$$\textbf{precision} = \frac{\text{True pos}}{\text{No. of predicted pos}} = \frac{\text{True pos}}{\text{True pos} + \text{False pos}} \tag{28}$$

$$\textbf{recall} = \frac{\text{True pos}}{\text{No. of actual pos}} = \frac{\text{True pos}}{\text{True pos} + \text{False neg}} \tag{29}$$

Convention: Define y=1 in presence of *rare class* that we want to detect.

11.4 Trading off Precision and Recall

Recall that for logistic regression, we have the hypothesis $0 \leq h_\theta(x) \leq 1$. Note that earlier we set the threshold to be 0.5. Now we can set the threshold higher (e.g, 0.8 for a more confident prediction), or lower (e.g. 0.3 for a greater coverage).

Consider two use cases:

1. Suppose we want to predict y=1 (cancer) only if we are very confident: We will set the threshold to be high.
   High precision, low recall.
2. Suppose we want to avoid missing too many cases of cancer (avoid false negatives): We will set the threshold to be low.
   High recall, low precision.

11.4.1 $\mathbf{F_1}$ score

Now that we have two metrics, precision and recall, we need a method to combine the two into a single metric evaluation for comparison purposes. An initial step may be to use the arithmetic mean. However, this can be inaccurate when either precision or recall is extremely high and the other extremely low. The mean will yield a somewhat high value. We need a better metric. Here we propose the F_1 score:

$$F_1 = 2\,\frac{PR}{P+R} \tag{30}$$

11.5 Large Data Rationale

At some point, we will need to acquire more data. A larger dataset is useful when $x \in \mathbb{R}^{n+1}$ has sufficient information to predict y accurately. A useful mental test is: given the input x, can a human expert confidently predict y?

When we use a learning algorithm with many parameters, e.g. logistic regression, linear regression with many features; neural network with many hidden units. This ensures a low bias algorithm. Therefore $J_{train}(\Theta)$ will be small.

Now, we can then use a large training set (which is unlikely to overfit) and ensure low variance. In such case, $J_{train}(\Theta) \approx J_{test}(\Theta)$, which makes the test error low.

12 Support Vector Machine

12.1 Optimization Objective

12.1.1 Alternative View of Logistic Regression

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

If y = 1, we want $h_\theta(x) \approx 1$, $\theta^T x >> 0$.
If y = 0, we want $h_\theta(x) \approx 0$, $\theta^T x << 0$
Cost of example:

$$\boxed{y\; log\; h_\theta(x) \;+\; (1-y)\; log(1-h_\theta(x))) \;=\; -y\; log\; \frac{1}{1\;+\;e^{-\theta^T x}} \;-\; (1-y)\; log\;(1-\frac{1}{1+\;e^{\theta^T\; x}})}$$

Consider the two terms in the above equation, where now we replace the cost with a straightened two-section curve:

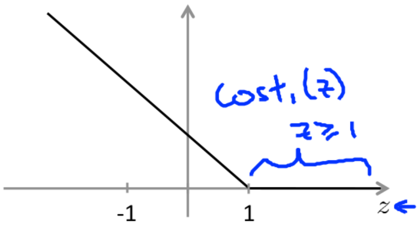


Figure 10: SVM cost_1

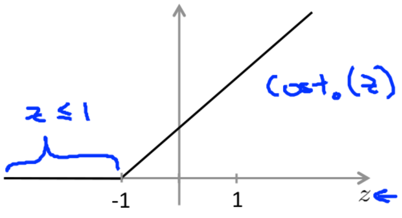


Figure 11: SVM cost_0

1. If y=1 , want $h_\theta(x) \approx 1$, $\theta^T x >> 0$. Here we label the curve as cost_0 (z), as shown in Figure 10.
2. If y = 0, we want $h_\theta(x) \approx 0$, $\theta^T x << 0$. Here we label the curve as cost_1(z), as shown in Figure 11..

12.1.2 Support Vector Machine

Recall logistic regression has the form as in Equation 19, shown below in an alternate form (taking the negative sign into the summation):

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \;\; - log\, h_\theta(x^{(i)}) \;\; - (1-y^{(i)})\, log\,(\,1-h_\theta(x^{(i)})\,)] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

The Support Vector Machine has a similar form as logistic regression, with a few conventions that differ from its logistic regression counterpart:

1. Remove scaling term ($\frac{1}{m}$). Note that this does not change the final result of minimized θ.
2. The logistic regression has the form: $A + \lambda B$; the SVM has the form $CA + B$. This is just a different way of parametrizing tradeoffs. Note here C behaves similarly to $\frac{1}{\lambda}$.

Therefore, the SVM hypothesis can be expressed in Equation 31.

$$\underset{\theta}{min}\, C \sum_{i=1}^{m} [y^{(i)}\, cost_1(\theta^T x^{(i)}) +\ (1 - y^{(i)})\, cost_0(\theta^T x^{(i)})] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 \tag{31}$$

The hypothesis

$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{if otherwise.} \end{cases}$$

12.2 Large Margin Intuition

Recall in Figure 10 and 11, we re-defined the function. Herein, we have shifted our threshold to z=1 and z=-1 respectively for predicting y = 1 and 0, respectively. This is different from previous threshold of 0 for logistic regression. We are more conservative in our predictions here, by using the 2 new segment cost functions, cost$_1(\theta^T x^{(i)})$ and cost$_0(\theta^T x^{(i)})$.

12.2.1 SVM Decision Boundary

Suppose we set C to be a large number, e.g. 10000. Under the minimization objective, we will be highly motivated to pick a θ such that the first term is close to zero, thus discarding the first term. The original equation will then be reduced effectively to

$$\underset{\theta}{min}\, \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

such that

$$\theta^T x^{(i)} \begin{cases} \geq 1 & \text{if } y^{(i)} = 1 \\ \leq -1 & \text{if } y^{(i)} = 0 \end{cases}$$

12.2.2 Large Margin Classifier

The Support Vector Machine is also known as the Large Margin Classifier. The word margin refers to the distance between the boundary and the data points. SVM will try to maximize the distance, as seen in Figure 12.

When C is too large (similar to a small λ), then we get decrease the power of the margin margin classifier, and the algorithm will fit too well to outliers.

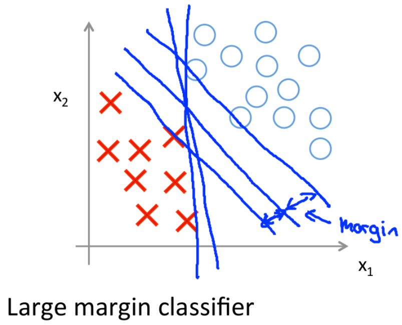


Figure 12: SVM Decision Boundary: Linearly Separable Case

12.3 Kernels

12.3.1 Non-linear decision boundary

For a non-linear decision boundary problem (as shown in Figure 13), we have:

$$h_\theta = \begin{cases} 1, & \text{if } \theta_0 x + \theta_1 x_1 + \dots \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

This can be written as

$$\theta_0 x + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \dots$$

, where

$$f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, f_4 = x_1^2, f_5 = x_2^2 \dots$$

However, high order polynomials are expensive. Is there a better choice of features f_i's?

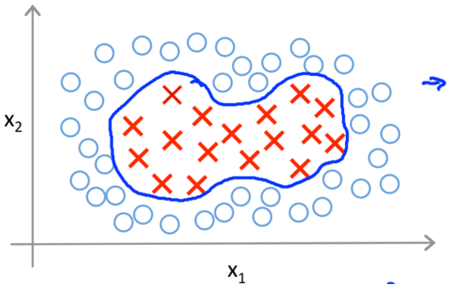


Figure 13: Nonlinear decision boundary problem

12.3.2 Kernel

Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$. This can be illustrated in Figure 14.

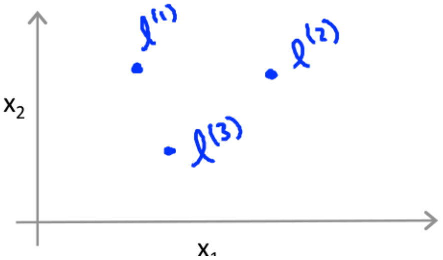


Figure 14: Kernel and landmark illustration

We then compute the features $f_i = kernel(x, l^{(i)})$ as follows:

$$f_i = \text{similarity}(x, l^{(i)}) = exp(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}) \tag{32}$$

This is a specific type of kernel, namely, the *Gaussian kernel.*

12.3.3 Kernels and Similarity

Note: The term $\|x - l^{(i)}\|$ in 32 can also be written as: $\sum_{j=1}^{n}(x^j - l_j^{(i)})^2$. We can analyze the term in two scenarios:

1. If $x \approx l^{(i)}$, then $\|x - l^{(i)}\| \approx 0$, consequently $f_i \approx 1$.
2. If x is far from $l^{(i)}$, then $\|x - l^{(i)}\|$ large, consequently $f_i \approx 0$.

Each landmark $l^{(i)}$ defines a new feature f_i, where the function approaches one when x is close to the landmark, and approaches zero otherwise.

The function can be visualized in Figure 15. Note that as the variance (σ) increases, f falls from the peak (1) slower.

12.3.4 Choosing Landmark

A natural question arises: *Where do we get $l^{(i)}$?*

Recall from 32, given x we can compute a feature ($\mathrm{f_i}$) as the similarity between the point and the landmark.

We predict y = 1, when

$$\sum_{i}^{n} \theta_i f_i \geq 0$$

In practise, θ_0 is chosen to be negative, where $\theta_i, (i \neq 0)$ are chosen to be some non-negative constants. As such, the closer x_i is to each l_i, f_i will tend to 1,

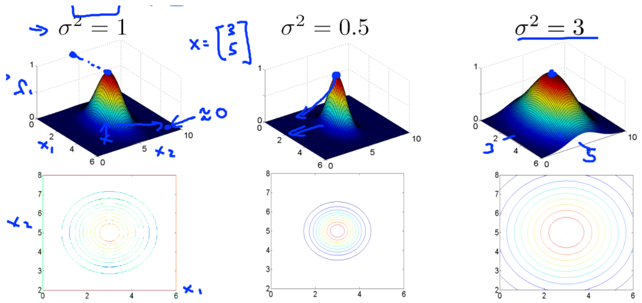
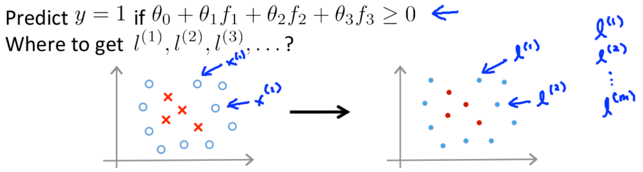


Figure 15: Guassian Kernel 3D plot

thus making the term $\theta_i f_i$ some positive number. We then take the sum of "weighted proximities to each landmark", and make the decision on y.

Observe Figure 16, where the markings of datasets we want is in blue, and red are the ones that we discard. Our goal is to guess what data belongs to our identification. Hence, we want to identify x that is close to our training examples.


Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \dots$?

Figure 16: Landmark and Training dataset

Comparing the above two paragraphs, it follows naturally that we should choose the training examples as our landmarks. Thus features are a measure of *proximity to training examples*, given an x.

12.3.5 SVM with Kernels

A summary of our setup:

- Given training examples: $(x^{(i)}, y^{(i)})$, i $= 1 \dots m$.
- Select landmarks: $l^{(i)} = x^{(i)}$
- Given example x: compute features $f_i = similarity(x, l^{(i)})$.
  For training example $(x^{(i)}, y^{(i)})$, where $x^{(i)} \in \mathbb{R}^{n+1}$ $(0 \dots n)$. We can compute the feature vector $f^{(i)} \in \mathbb{R}^{m+1}$, where m is the number of landmarks

(training examples).

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_i^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$$

1. $f_0^{(i)} = 1$ by default.
2. $f_i^{(i)} = 1$ by design (check Equation 32; similarity of $x^{(i)}$ to $l^{(i)} = x^{(i)}$ is 1).

Formally:

1. **Hypothesis**: Given $x \in \mathbb{R}^{n+1}$, compute features $f \in \mathbb{R}^{m+1}$.
   Predict "y = 1", if $\theta^T f = \sum_{i=0}^{m} \theta_i f_i \geq 0$
2. **Training**:

   $$\min_\theta C \sum_{i=1}^{m} [y^{(i)}\, cost_1(\theta^T f^{(i)}) + \ (1 - y^{(i)})\, cost_0(\theta^T f^{(i)})] + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

   Remarks:

   - The above equation is similar to 31.
   - We calculate the cost based on θ and f^i.
   - In the second summation, $n = m$; number of features=number of training sets.
3. **Implementation Note**

   - $\sum_{j=1}^{m} \theta_j^2 = \theta^T \theta; \theta = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$
   - Can also scale θ, i.e. $\theta^T M \theta$.
   - M increases computation efficiency.

12.3.6 SVM Parameters

We have two parameters: C and λ:

C$= \frac{1}{\lambda}$:

1. Large C (small λ): lower bias, higher variance.
2. Small C (large λ): Higher bias, low variance.

σ^2: (see Figure 15)

1. Large σ^2: (features f_i vary more smoothly) higher bias, lower variance.
2. Small σ^2: (changes abruptly) lower bias, higher variance.

12.4 Using an SVM

- Use SVM software package (e.g. linlinear, libsvm, ...) to sole for parameters θ
- Specification:
  1. Parameter C.
  2. Kernel (similarity function)
     - No kernel ("linear kernel"): predicts "y=1" if $\theta^T x \geq 0$.
     - Guassian kernel: $f_i = exp(-\frac{\|x-l^{(i)}\|^2}{2\sigma^2})$, need to choose σ^2, perform feature scaling.
- Other choices of kernel: not all similarity functions make valid kernels. Need to satisfy *Mercer's Theorem* for convergence. Off-the-shelf kernels:
  - Polynomial kernel
  - String kernels (text)
  - Chi-squared kernel
  - Histogram intersection kernel

12.5 Multiclass Classification

For illustration, see Figure 17. Many SVM packages have built-in functionalities. Otherwise, we can implement using one-vs-all method:

- Train K SVMs, each distinguish $y = i, \forall i = 1, 2, \ldots K$ from the rest.
- Get $\theta^{(1)}$, $\theta^{(2)}$, $\theta^{(3)}$, ..., $\theta^{(K)}$.
- Pick class i with largest $(\theta^{(i)})^T x$.

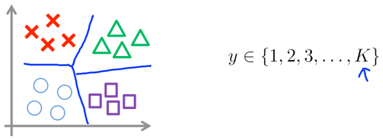


Figure 17: Multiclass SVM

12.6 Logistic Regression v.s. SVMs

Let:

- n = number of features ($x \in \mathbb{R}^{n+1}$)
- m = number of training examples.

1. If $n \gg m$: use logistic regression, or SVM without a kernel (linear kernel).
2. If $n < m$: use SVM with Guassian Kernel.
3. IF $n \ll m$: Create/add more features, then use logistic regression or SVM without a kernel.
4. Neural network: likely to work in all cases, but may be slower to train.

13 Clustering

13.1 Unsupervised learning introduction

Sometimes we do not have a training set with defined labels (results); therefore, we need to find some structure from the training set. **Clustering** is one way to do so, by grouping close data points together.

Applications of clustering:

- Market segmentation.
- Social network analysis.
- Organize computer clusters.
- Astronomical data analysis.

13.2 K-means algorithm

K-means algorithm performs clustering by defining a "centroid" for each cluster and improve the accuracy of centroids through minimizing the total squared distances from the centroid to all points in the particular cluster.

1. Input:

- K (number of clusters).
- Training set $x^{(1)}, x^{(2)}, x^{(3)}, \dots x^{(m)})$. $x^{(i)} \in \mathbb{R}^n$ (We drop the $x_0 = 1$ convention.

2. **Algorithm** The K-means algorithm has three main steps:

   (a) **Initialization**: Initialize cluster centroids (random).

   (b) **Cluster Assignment**: (Loop: for all data points x) assignment of each data point into a cluster, depending on the proximity to the cluster centroid.

   (c) **Move Centroid**: (Loop: for all clusters) refine the centroid of each cluster by taking the mean location of all data points for an individual cluster.

K-means Algorithm

```
Randomly initialize $K$ cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$
Repeat {
    for i=1 to m do
        $c^{(i)} := k$, s.t. $\min_k \|x^{(i)} - \mu_k\|^2$
        {Cluster assignment: index $(1 \dots K)$ of cluster centroid closest to $x^{(i)}$}
    end for
    for k= 1 to K do
        $\mu_k :=$ mean of points assigned to cluster k
        {Move centroid}
    end for
}
```

3. K-means for non-separate clusters: sometimes the clusters are not well defined (separated enough). The algorithm will still produce clustering based on the proximity. An example is the **T-shirt sizing problem** (See Figure 18) : for a given set of weight-height distribution from market survey, how many sizes should the manufacturer produced (S, M, L) or (XS, S, M, L, XL)?

13.3 Optimization Objective

1. $c^{(i)}$ = **index**of cluster (1, 2, $\dots K$) to which example $x^{(i)}$ is currently assigned.

2. μ_k = cluster centroid k. [Coordinate $\mu_k \in \mathbb{R}^n$]

3. $\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned.

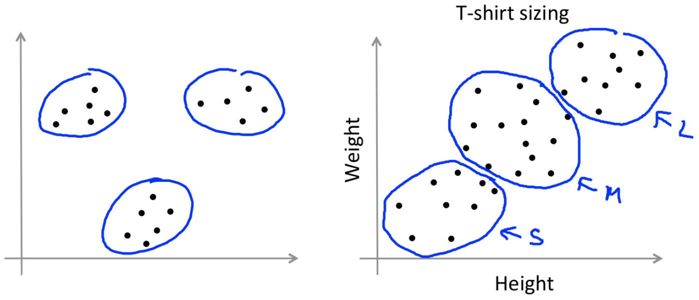


Figure 18: Non-separated clusters: T-shirt sizing

4. **Optimization objective**:

$$\min_{\{c^{(i)}\},\{\mu_k\}} J(\{c^{(i)}\},\{\mu_k\}) \tag{33}$$

$$J(\{c^{(i)}\},\{\mu_k\}) = \frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2 \tag{34}$$

Note:

(a) The first for loop in the algorithm (i) minimizes $J(\dots)$ with respect to $c^{(i)}$ and holds μ_k constant.

(b) The second for loop in the algorithm (k) minimizes $J(\dots)$ with respect to μ_k and holds $c^{(i)}$ constant.

13.4 Random Initialization

Rules:

1. $K <$ m (number of training examples).
2. Randomly pick K training examples.
3. Set μ_k to the K examples chosen in the previous step.

There might be different clustering based on different initial random selection. Therefore one can run the K-means algorithm multiple times (100), each with a different random initialization (selection of examples as initial centroids). Compute all cost function (distortion), then pick the one that gives the lowest cost J.

13.5 Choosing the number of clusters (K)

Sometimes it is ambiguous how many clusters there are, e.g. could be 2 or 4. One method of determining the number of clusters is the *Elbow method.* We compute the cost and vary K- we will usually see a sharp drop in cost at the star, and the decreasing rate tends to slow down after the "elbow points". The elbow point is the point where the cost function's slopes magnitude starts decreasing.

However, not all scenario will produce an obvious "elbow" (refer to the graph on the right in Figure 19). In such cases, the Elbow method doesn't give much insight.

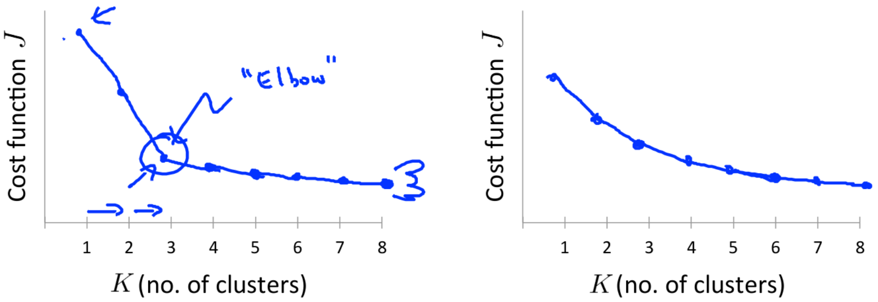


Figure 19: Elbow method in apparent cases (left) and non-apparent cases(right)

In some cases, there will only be some definitive Ks to choose from. This is often seen in K-means based on a metric for some downstream purpose. Recall the T-shirt sizing problem, in such cases, we have standard numbers of clusters (3, 5, 7, etc.). One can then iterate through all possible options, and proceed with the one which yields the lowest cost.

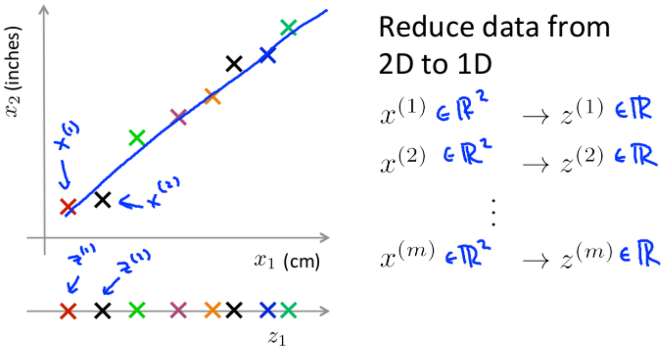
14 Dimensionality Reduction

14.1 Motivation

14.1.1 Data Compression

Sometimes there exists redundant data dimensions, e.g. repeated quantities in different units. In general, such dimensions are correlated by some hyperbolic surfaces . In the 2D case, two dimensions x_1, x_2 can be compressed into a line that describes the correlation of the two dimensions; thus we need only the information of that "line" and where each pair of (x_1, x_2) lies on that line (See Figure 20). This can further be generalized into more dimensions, e.g. 3 dimensions lying on a 2D-plane (Figure 21).

We compress the data by projecting the data points on to the plane of correlation approximation and re-coordinate into z. This reduces the data to one-less dimension.



Reduce data from 2D to 1D

$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$

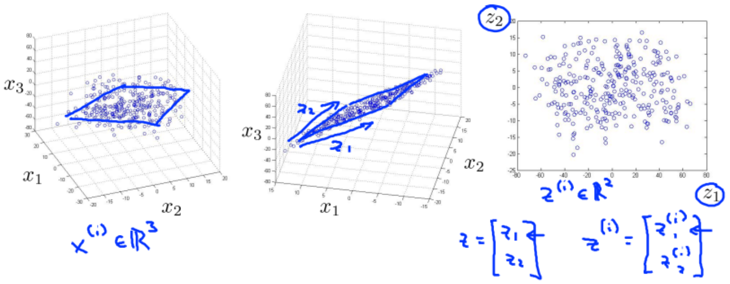
$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$

$\vdots$

$x^{(m)} \in \mathbb{R}^2 \rightarrow z^{(m)} \in \mathbb{R}$

Figure 20: Data compression from 2D to 1D

Reduce data from 3D to 2D



$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$ $z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$

Figure 21: Data compression from 3D to 23D

14.1.2 Data Visualization

For visualization purposes, we often wish to group the dimensions into two or three dimensions, of which visualizations are easier perceived by human.

14.2 Principal Component Analysis

14.2.1 Problem Formulation: n-d to k-d

The projection error is the orthogonal distance squared from the data to that projected on the plane. To find the plane of correlation approximation, we need to find k vectors: $u^{(1)}, \ldots, u^{(k)}$ onto which the data projects, so as to minimize the projection error. We want to project the data onto the linear subspace span by the k vectors.

For example: from 2D to 1D, we need to find a direction vector ($u^{(1)} \in \mathbb{R}^n$).

Note: **linear regression is different than PCA.** The former minimizes the squared error ($|y-h(x)|^2$) (error bar is parallel to the y-axis, not the shortest distance); the latter minimizes the orthogonal distance from the point to the projection. See Figure 22 for the comparison.

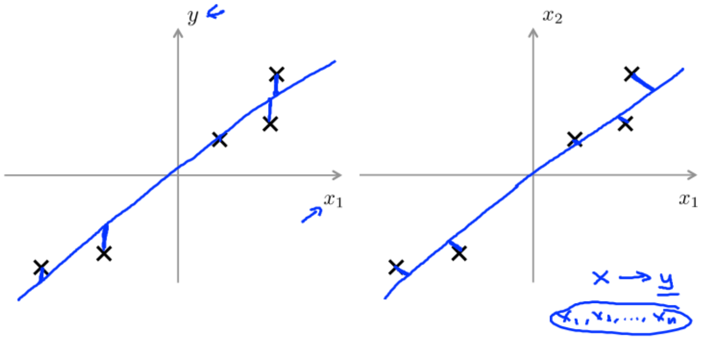


Figure 22: Linear regression vs PCA. Linear regression minimizes the error of y to the prediction. PCA minimizes the error/distance from the original point orthogonal to the projected

14.3 Principal Component Analysis: Algorithm

14.3.1 Data Preprocessing

- Training set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$
- Preprocessing (feature scaling and mean normalization):

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

14.3.2 Principal Component Analysis

Reduce data from n-dimensions to k-dimensions

- Covariance matrix:

$$\Sigma = \frac{1}{m} \sum_{i=1}^{n} [x^{(i)}][x^{(i)}]^T = \frac{1}{m} X^T X$$

  Recall from Equation 6, the design matrix X has each example in a row.

- Eigenvectors of Σ: `[U, S, V] = svd{Sigma}`
  The svd function stands for "single value decomposition". The U matrix is matrix of eigenvectors in columns:

$$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

- Projection: We can take the first k columns from the eigenmatrix U to form U_{reduce}:

$$U_{reduce} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times k}$$

  Perform the projection:

$$z^{(i)} = U_{reduce}^T \cdot x^{(i)} \in \mathbb{R}^{k \times 1} \tag{35}$$

14.4 Reconstruction from compressed representation

We can guess the original data by inverting 35.

$$x \approx x_{approx} = U_{reduce} \cdot z$$

14.5 Choosing k (num of principal components)

- Average squared projection error: $\frac{1}{m} \sum_{i=1}^{n} \|x^{(i)} - x_{approx}^{(i)}\|^2$
- Total variation in data: $\frac{1}{m} \sum_{i=1}^{n} \|x^{(i)}\|^2$
- Choose k s.t. "99% variance is retained":

$$\frac{\frac{1}{m} \sum_{i=1}^{n} \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^{n} \|x^{(i)}\|^2} \leq 0.01 \tag{36}$$

- Procedure: Try PCA with k=1, compute U_{reduce}, check Equation 36, $k \leftarrow k+1$.

- Implementation note: Equation 36 can be solved differently using the S matrix. $S \in \mathbb{R}^{n \times n}$ is a diagonal matrix. For a given k, we can compute Equation 36 by:
$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}}$$
, or alternatively,
$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

14.6 Advice for applying PCA

1. Supervised learning speed-up: define the mapping $x^{(i)} to z^{(i)}$ by running PCA on the **training set**. This mapping can be later used on cross validation and test sets.

2. Application of PCA

   (a) Compression (reduce space), speed up learning algorithm.

   (b) Visualization.

   (c) **Bad usage**: prevent overfitting (fewer features, less likely to overfit). However, PCA discards certain information through the projection step. It is better to use regularization (λ)

3. Design ML system without PCA first, if that doesn't work, then try PCA.

15 Anomaly Detection

15.1 Motivation: Density Estimation

Generate a function *p(x)*, such that $p(x) < \epsilon$ suggest anomaly. The crux is to flag unusual behaviour, for example:

- Fraud detection
- Manufacturing
- Computer load in data center

15.1.1 Guassian distribution

$$x \sim \mathcal{N}(\mu, \sigma^2) \tag{37}$$

- μ: mean
- σ^2: variance

15.2 Algorithm

1. Choose features x_i that are indicative of anomalous examples.

2. Fit parameters

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_i \\ \vdots \\ \mu_n \end{bmatrix} = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)} \tag{38}$$

and

$$\boldsymbol{\sigma^2} = \begin{bmatrix} \sigma_1^2 \\ \vdots \\ \sigma_i^2 \\ \vdots \\ \sigma_n^2 \end{bmatrix} = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2 \tag{39}$$

3. Given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} exp(\frac{-(x_j - \mu_j)^2}{2\mu_j^2}) \tag{40}$$

Anomaly if $p(x) < \epsilon$

15.3 Developing and Evaluating an Anomaly Detection System

15.3.1 Developing

In a typical anomaly detection setting, we have a large number of anomalous examples, and a relatively small number of normal/non-anomalous examples. Dividing labelled data:

- Training set: 60% of normal (y=0); no anomalies.
- Cross validation set: 20% of normal (y=0); 50% anomalies (y=1).
- Test set: 20% of normal (y=0); 50% anomalies (y=1).

15.3.2 Evaluation

- Real-number evaluation
- Evaluation metrics:
  - True positive, false positive, false negative, true negative

  - Precision/Recall
  - F_1 score

- If no labelled data (or all data labelled as y=0), it is still possible to learn $p(x)$, but harder to evaluate the system or choose a good value of ϵ.

15.4 Anomaly Detection vs Supervised Learning

Refer to Table 8.

Anomaly detection	Supervised learning
Small number of positive examples [y=1] (0-20) and large number of negative examples [y=0]	Large number of positive and negative examples
Many different types of anomalies. Hard to learn anomalies from positive examples.	Enough positive examples to learn positivity.
Future anomalies may look nothing like previous anomalies	Future positive examples similar to training set.

Table 8: Anomaly detection vs Supervised learning

15.5 Choosing what features to use

- **Goal**: Want $p(x)$ large for normal examples x; $p(x)$ small for anomalous examples x.
- Ideally require Guassian features.
- If not Guassian, require preprocessing, e.g. log(), $\sqrt{x}$, etc.
- Most common problem: $p(x)$ is comparable for both normal and anomalous examples → come up with new features (e.g. ratios of current fetatures.)

15.6 Multivariate Guassian Distribution

$x \in \mathbb{R}^n$. Model $p(x)$ with all dimensions in tandem. Parameters: $\boldsymbol{\mu} \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance matrix).

15.6.1 Anomaly detection using multivariate Guassian distribution

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)) \tag{41}$$

where

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \tag{42}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T \tag{43}$$

15.6.2 Original model vs Multivariate Guassian

Refer to Table 9.

Original model	Multivariate Guassian
$\prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$ (Eq. 40)	Eq. 41
Manually combine features to capture anomalies	Automatically captures correlations between features.
Computationally cheaper and scales well	Computationally expensive (matrix)
Works on small training set size (m)	Must have training set size (m) > features size (n), i/e/ m *geq* 10n for Σ to be invertible.

Table 9: Anomaly detection: original model vs multivariate Guassian

16 Recommender Systems

16.1 Problem formulation

1. X = $\begin{bmatrix} -\left(x^{(1)}\right)^T- \\ -\left(x^{(2)}\right)^T- \\ \vdots \\ -\left(x^{(n_m)}\right)^T- \end{bmatrix}$ For each movie i ($i = 1 \ldots n_m$), form a feature vector $x^{(i)}$ to characterize the movie.

2. $\Theta = \begin{bmatrix} -\left(\theta^{(1)}\right)^T- \\ -\left(\theta^{(2)}\right)^T- \\ \vdots \\ -\left(\theta^{(n_u)}\right)^T- \end{bmatrix}$ For each use j ($j = 1 \ldots n_u$), there is a parameter vector $\theta^{(j)}$ that characterizes the user's interest, such that $(\theta^{(j)})^T x^{(i)}$ is a prediction of user j's rating on movie i.

3. $Y = \{\ y(i,j)\ \}$, where $y(i,j)$ is the actual ratings of use j on movie i.

4. $R = \{\ r(i,j)\ \}$, where $r(i,j)$ is the "valid" bit for $y(i,j)$, i.e. if user i gave a rating on movie i.

16.2 Collaborative filtering

The cost function for collaborative filtering with regularization is:

$$J(X,\Theta) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \left(\frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2\right) + \left(\frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2\right) \tag{44}$$

One can then apply gradient descent:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} \frac{\partial}{\partial\theta_k^{(j)}} J(X,\Theta) \tag{45}$$

$$x_k^{(i)} := x_k^{(i)} - \alpha \sum_{i:r(i,j)=1} \frac{\partial}{\partial x_k^{(j)}} J(X,\Theta) \tag{46}$$

where

$$\frac{\partial J}{\partial x_k^{(i)}} = \sum_{j:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)\theta_k^{(j)} + \lambda x_k^{(i)} \tag{47}$$

$$\frac{\partial J}{\partial \theta_k^{(j)}} = \sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)x_k^{(i)} + \lambda \theta_k^{(j)} \tag{48}$$

The collaborative filtering algorithm is:

1. Initialize $x^{(i)}, \theta^{(j)}$ to small random values.
2. Minimize $J(X,\Theta)$ using gradient descent.
3. Predict user i rating on movie j via $(\theta^{(j)})^T x^{(i)}$

17 Gradient Descent with Large Datasets

17.1 Stochastic gradient descent

- **Batch gradient decent**: In each iteration, sum cost of all m examples, then taking one step of minimization (improving parameters). Might be expensive as number (of training examples) scales up.
- **Stochastic gradient descent**: take single cost of a training example, start improving the parameters towards global minimum. The algorithm may not converge immediately in a direct path; the route taken might be conservative and biased towards each individual sample. Might not converge to a single minimum; algorithm might end up wandering near the minimum– this is a "good enough" hypothesis in most cases. Usually 1-10 iterations.
- **Mini-batch gradient descent**: use b examples in each iteration (rather than all m). Step in size of b. Also faster than Batch gradient descent. Outperforms Stochastic gradient descent if good vectorization can be performed (achieve parallelism). Downside: extra variable b.

17.2 Stochastic gradient descent convergence and tuning

Recall for Batch gradient descent, we had to plot $J_{train}(\theta)$ as a function of number of iterations of gradient descent.

For Stochastic gradient descent, during learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$. For every 1000 iterations, plot averaged over last 1000 examples. This gives a running estimate of how well minimization performs. IF we see that the running estimate increases over time, we can try to use a smaller learning rate α; each iteration of the stochastic gradient descent will take a smaller step, thereby increasing the likelihood of convergence.

17.3 Advanced topic: online learning

- Take one example, improve parameters based on the single example (this is essentially stochastic gradient descent). Discard example.
- Useful when website gets a stream of data from different users.
- Can adapt to varying user preference.
- Search query keywords (Predicted CTR Click-through-rate): $p(y = 1|x; \theta)$.

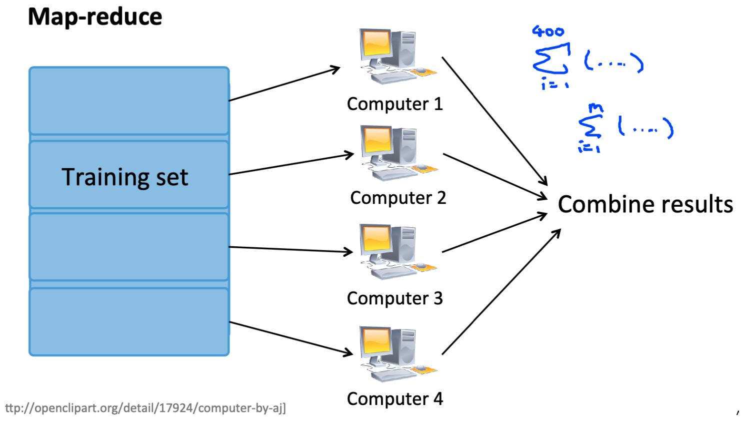
17.4 Advanced topic: map reduce and data parallelism

- Split examples into batches for different machines.
- Each machine will compute semi-total cost according to the batch it was assigned.
- Combine calculated cost to perform gradient descent and get a single θ.
- Net result is equivalent to original
- Applicable to multi-core machines: no network latency.

17.5 Stochastic gradient descent applications

Many learning algorithms can be expressed as computing sums of functions over the training set; map-reduce is good candidate for such algorithms, where the bottleneck stems from summing over the training set. Because of network latency and other overhead, if we run map-reduce on N computers, we might get less than an N-fold speedup.

- Batch gradient descent: split dataset into N smaller batches, compute the gradient for each smaller batch on each machine, then average the gradients on a central computer for the gradient update. This works for regressions and neural network training



ttp://openclipart.org/detail/17924/computer-by-aj]

Figure 23: Map reduce

- Stochastic gradient descent: **Not applicable**. Since stochastic gradient descent processes one example at a time and performs update after each example, it cannot be easily parallelized.